Filter content items (examples)

October 6, 2021 • Jan Cerman • 4 min read • TypeScript

When building your apps, you'll use the Delivery API to get content items. The Delivery API provides several filters to help you refine your requests based on your requirements.

What do you know about the content you want to get? Find a relevant scenario below and adjust the code to get the content you need.

Item ID or codename

To get an item based on its identifier (like ID or codename), use the `equals` filter. For example, if you need to get an item by its internal ID, you match the item's ID (`system.id`) against the ID you have.

TypeScript

```
// Gets an item by its internal ID
deliveryClient.items<ContentItem>()
  .equalsFilter('system.id', '2f7288a1-cfc8-47be-9bf1-b1d312f7da18')
  .toObservable()
  .subscribe(response => console.log(response));
```

If you have multiple identifiers of a single type (for example, only IDs or codenames) and want to retrieve them in a single request, use the `in` filter.

TypeScript

```
// Gets three items by their codenames. The codenames are unique per project.
deliveryClient.items<ContentItem>()
  .inFilter('system.codename', ['delivery_api', 'get_content', 'hello_world'])
  .toObservable()
  .subscribe(response => console.log(response));
```

If you know the item's `codename` , retrieve the content item directly for better performance.

Content types

To get items based on a single content type, specify the content type's codename (`system.type`) using the `equals` filter.

TypeScript

```
// Gets items based on the type Product
deliveryClient.items<ContentItem>()
  .equalsFilter('system.type', 'product') // Same as using .type('product')
  .toObservable()
  .subscribe(response => console.log(response));
```

To get items based on multiple content types, specify the content types' codenames using the `in` filter.

TypeScript

```typescript
// Gets items based on the types Product, Article, and News
deliveryClient.items<ContentItem>()
  .inFilter('system.type', ['product', 'article', 'news'])
  .toObservable()
  .subscribe(response => console.log(response));
```

Date & time and numbers

To get items by a datetime value, you can use one of the comparison filters. This lets you retrieve content modified or released before or after a certain date. The following code shows the filters used on the last content modification date and a date & time element.

TypeScript

```typescript
// Note: Date & time element values are provided by users and stored with minute precision. The system.last_modified value reflects Last content change to an item and is stored with ms precision.
// Gets items modified after April 9 2020, 9 am UTC+0
deliveryClient.items<ContentItem>()
  .greaterThanFilter('system.last_modified', '2020-05-09T09:00:00.000000Z')
  .toObservable()
  .subscribe(response => console.log(response));

// Gets items released at or after April 9 2020, 7 am UTC+0
deliveryClient.items<ContentItem>()
  .greaterThanOrEqualFilter('elements.release_date', '2020-05-09T07:00:00Z')
  .toObservable()
  .subscribe(response => console.log(response));

// Gets items modified before April 5 2020 UTC+0. Last match would be at 2020-05-04T23:59:59.
deliveryClient.items<ContentItem>()
  .lessThanFilter('system.last_modified', '2020-05-05')
  .toObservable()
  .subscribe(response => console.log(response));

// Gets items released at or before April 5 2020 10:30 am UTC+0
deliveryClient.items<ContentItem>()
  .lessThanOrEqualFilter('elements.release_date', '2020-05-05T10:30:00Z')
  .toObservable()
  .subscribe(response => console.log(response));
```

Use the same approach for number elements.

Range of dates and numbers

To get items based on a date range, you need to specify two datetime values using the `range` filter.

TypeScript

```typescript
// Note: Date & time element values are provided by users and stored with minute precision. The system.last_modified value reflects Last content change to an item and is stored with ms precision.
// Gets items modified between April 5, 2020 10:30 UTC and April 7, 2020, 7:00 UTC
deliveryClient.items<ContentItem>()
  .rangeFilter('system.last_modified', '2020-05-05T10:30:00', '2020-05-07T07:00:00')
  .toObservable()
  .subscribe(response => console.log(response));
```

To get items based on a number range, you need to specify two numbers. The numbers can be either integers like `3` or floats like `3.14`.

TypeScript

```typescript
// Gets items whose rating is at least 6.5 and at most 9
deliveryClient.items<ContentItem>()
  .rangeFilter('elements.product_rating', '6.5', '9')
  .toObservable()
  .subscribe(response => console.log(response));
```

Text and rich text

To get items based on the value of a text or rich text element, you need to specify the value using the `equals` filter. The same approach also applies to custom elements.

TypeScript

```typescript
// Gets items whose Title element value equals to 'Hello World'
deliveryClient.items<ContentItem>()
  .equalsFilter('elements.title', 'Hello World')
  .toObservable()
  .subscribe(response => console.log(response));
```

Taxonomy and multiple choice

To get items tagged with specific terms, you need to specify the terms using the `contains`, `any`, or `all` filters.

TypeScript

```typescript
// Note: Filters work with codenames of the tags.
// Gets items tagged with one specific tag
deliveryClient.items<ContentItem>()
  .containsFilter('elements.tags', ['kontent'])
  .toObservable()
  .subscribe(response => console.log(response));
```

```
// Gets items tagged with a list of specific tags
deliveryClient.items<ContentItem>()
  .allFilter('elements.tags', ['kontent', 'headless'])
  .toObservable()
  .subscribe(response => console.log(response));

// Gets items tagged with at least one tag from the list
deliveryClient.items<ContentItem>()
  .anyFilter('elements.tags', ['football', 'soccer'])
  .toObservable()
  .subscribe(response => console.log(response));
```

Also works for multiple choice and custom elements

Use the same approach to get items based on a specific value or values of multiple choice element.

For custom elements, the contains, any, and all filters work only if the element's value is a stringified array of strings. For example, `"[\"DE\",\"US\",\"UK\"]"` .

URL slug

The URL slug element element value is stored in the same way as text or rich text. This means the approach to get items by a specific URL slug is the same, using the `equals` filter.

TypeScript

```
// Gets items whose URL slug equals to sample-url-slug
deliveryClient.items<ContentItem>()
  .equalsFilter('elements.url_slug', 'sample-url-slug')
  .toObservable()
  .subscribe(response => console.log(response));
```

To ensure you get content in a specific language, use the `language` parameter in your requests.

Author of an article and other relationships

When you link multiple items together you might want to retrieve your items based on their relationships.

For example, you can have several articles written by a single author, Jane. Each article has a linked items element named *Author*. This element contains a reference to a content item that represents the author Jane. The reference in the *Author* element is stored as a codename of the *Jane* content item.

TypeScript

```
// Gets items attributed to Jane.
deliveryClient.items<ContentItem>()
  .containsFilter('elements.author', ['jane_doe'])
  .toObservable()
  .subscribe(response => console.log(response));

// Gets items attributed to at least Jane, John, or both.
```

```
deliveryClient.items<ContentItem>()
  .anyFilter('elements.author', ['jane_doe', 'john_wick'])
  .toObservable()
  .subscribe(response => console.log(response));
```

Subpages

When using Web Spotlight to manage your website content, you can retrieve subpages the same way as your linked items.

For example, you can have various insurance-related pages linked in multiple places on your website. To get the pages that reference the *Travel insurance* page in a subpages element called *Subpages*, use the `contains` filter in your request.

TypeScript

```
// Gets pages linking travel insurance as their subpage.
deliveryClient.items<ContentItem>()
  .containsFilter('elements.subpages', ['travel_insurance'])
  .toObservable()
  .subscribe(response => console.log(response));

// Gets pages linking at least travel insurance, car insurance, or both as their subpage.
deliveryClient.items<ContentItem>()
  .anyFilter('elements.subpages', ['travel_insurance', 'car_insurance'])
  .toObservable()
  .subscribe(response => console.log(response));
```

What's next?

- Build your app on solid foundations and best practices on getting content.
- When you get the right items, it's time to resolve and render your content.
- Take a deep dive into the Delivery API filters in the API reference.